基于深度聚类的开源软件漏洞检测方法 *

李元诚 [1]，黄　戎 [1]，来风刚 [2]，毛一凡 [2]，蔡力军 [3]

(1. 华北电力大学 控制与计算机工程学院，北京 102206; 2. 国家电网公司信息通信分公司，北京 100761; 3. 国网福建省电力有限公司信息通信分公司，福州 350003)

摘　要：近年来开源软件频频爆出高危漏洞，对企业信息系统安全造成巨大威胁。针对开源软件漏洞，提出一种基于深度聚类算法的软件源代码漏洞检测方法。该方法利用代码图模型构造开源软件代码属性图，遍历得到关键代码节点并提取出应用程序编程接口（API）序列，并将其嵌入向量空间，以关键代码为中心进行聚类，根据聚类结果计算每个函数的异常值，生成检测报告并匹配漏洞库，从而检测出源代码中的漏洞。实验结果表明，该方法能够定位开源软件中漏洞所在的关键代码段并检测出相应漏洞。

关键词：开源软件；漏洞检测；源代码分析；深度学习；聚类

中图分类号：TP　　doi: 10.19734/j.issn.1001-3695.2018.09.0721

Open source software vulnerability detection method based on deep clustering

Li Yuancheng[1], Huang Rong[1], Lai Fenggang[2], Mao Yifan[2], Cai Lijun[3]

(1. *School of Control & Computer Engineering*, *North China Electric Power University*, *Beijing* 102206, *China*; 2. *State Grid Information & Telecommunication Branch*, *Beijing* 100761, *China*; 3. *State Grid Fujian Information & Telecommunication Branch*, *Fuzhou* 350003, *China*)

Abstract: In recent years, open source software has frequently exposed high-risk vulnerabilities, posing a huge threat to the security of enterprise information system. Aiming at the open source software vulnerability, this paper proposed a software source code vulnerability detection method based on deep clustering algorithm. This method uses code graph model to construct the code attribute map and traverses the key code nodes to extract the application programming interfaces (API) sequence, then takes the key sequence as the center to cluster and calculates the outliers of the function in each clustering to generate a test report, matches the vulnerability library to detect vulnerabilities in the source code. The experimental results show that the proposed method can locate the key code segments of the vulnerability in open source software and detect the vulnerability.

Key words: open source software; vulnerability detection; source code analysis; deep learning; clustering

0　引言

随着互联网和大数据技术的高速发展、社会信息化程度不断提高，开源软件由于具有开放、共享、商用免费、功能灵活等特点，不断融入到各行各业的信息化建设中。开源软件是指允许用户基于 OSI 列出的开源协议,在协议许可的范围内自由使用、修改软件源代码,且可以将开源代码与其他软件代码进行结合使用的一种软件形式。开源软件的存在提高了软件的开放性和行业的开发水平，促进了标准化和软件开发的良性循环。由于开源软件的优势，企业在自身信息化建设中，不断扩大对开源软件的使用范围，以节约开发成本，提高企业效益。目前，开源软件在系统框架，日志，数据存储、处理以及传输等信息系统重要组成部分得到不同程度的使用。虽然开源软件的代码质量总体上在不断提高，但是开源软件的安全问题仍然普遍存在，导致企业面临大量的安全隐患和风险。据报导，来自开源组件和源码的安全威胁呈现出几何级数增长，频频爆出高危漏洞，例如 Struts2、open SSL 等。因此，为了保证企业信息系统的安全可靠，需要对开源软件进行安全检测。

漏洞是软件安全的重点研究课题[1~4]，随着机器学习的不断发展，利用机器学习算法来挖掘分析软件漏洞受到关注，通过代码度量[5,6]，图模型[7,8]等方法，提取代码特征，采用支持向量机（SVM）、随机森林（RF）、邻近等算法识别开源软件漏洞。此外，Shar 等人[9]基于数据流分析提取静态代码属性作为机器学习特征来识别开源软件中的漏洞。Scandariato、Pang 等人[10,11]引入自然文法语言挖掘技术，如词袋（bag-of-word）技术，N-gram 语言模型等，提取代码特征，并使用分类模型来识别软件中的漏洞。

采用机器学习检测漏洞，需要进行有效的建模，同时特征要包括语法、控制流、数据流等信息。然而，传统度量标准不适用于软件漏洞识别，需要定义特定的度量标准。由于开源软件漏洞通常来源于软件本身的设计缺陷、编程时编写错误、交互处理中的缺陷以及逻辑缺陷，并与特定的 API 序列有联系，因此可通过源代码中关键代码片段的 API 序列来检查开源软件漏洞。本文通过控制流和数据流构建源代码的属性图，提取出关键代码段的 API 序列，采用深度聚类算法分[12]析量化后的 API 序列，计算序列异常值以检测出开源软件中漏洞。

收稿日期：2018-09-06；**修回日期：**2018-11-15　**基金项目：**国家电网公司总部科技项目（SGFJXT00YJJS1800074）

作者简介：李元诚（1970-），男，教授，博导，博士，主要研究方向为机器学习与信息安全（yuancheng@ncepu.edu.cn）；黄戎（1995-），女，硕士研究生，主要研究方向为机器学习与信息安全；来风刚（1971-），男，高级工程师，硕士，主要研究方法为计算机科学与技术；毛一凡（1964-），男，工程师，主要研究方向为计算机系统与软件安全；蔡力军（1964-），男，高级工程师，主要研究方向为信息安全、通信与信息系统.

1 开源软件源代码特征提取

1.1 源代码属性图

源代码图模型是程序编译过程中形成的中间表示，包括抽象语法树，控制流程图，程序依赖图等。抽象语法树展示了程序的嵌套结构；控制流程图表明了程序中语句的执行顺序；程序依赖图显示了数据流和控制流的走向。传统的代码图模型生成方法需要搭建一定的编译环境，并找到适合该代码编程语言的编译工具以及所有的头文件。对于复杂、陈旧的程序，使用该方法生成代码图模型的过程十分复杂，并且可能会出现头文件缺失等问题。因此，本文利用开源工具Joern[13]从源代码中抽取出抽象语法树，控制流程图以及程序依赖图，并组合成生成代码属性图。Joern 是一个利用 Island Grammar，并基于分析器生成工具 ANTLR 构建的分析器。Island Grammar 是一种描述底层语言的语法，可以在不检查文本语法的情况下进行语言分析。ANTLR 是一个可以根据输入自动化生成语法树并可视化展示的开源语法分析器。

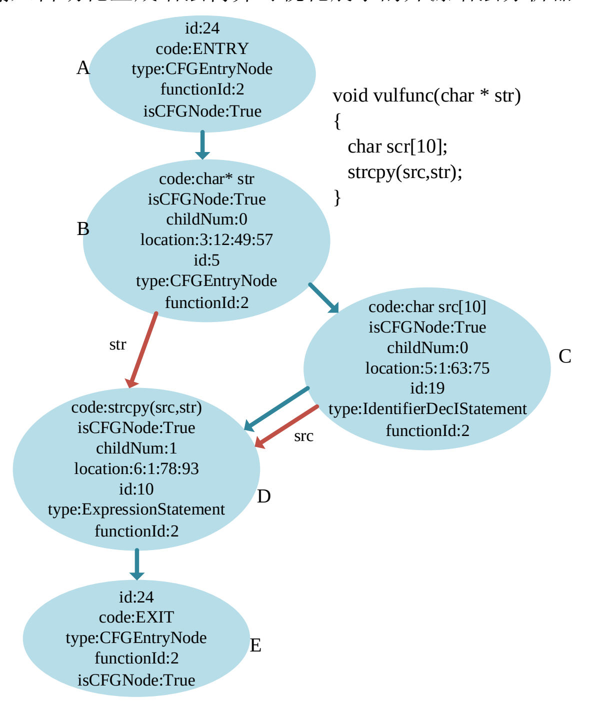


图 1 示例代码属性图

Fig. 1 Attribute map of sample code

在形式上，代码属性图是一个带有边缘标记的归属多图[14]，可以在图的节点和边上存储数据来标记节点之间的关系。对装入程序的每一个函数都生成一个代码属性图，图中包含该函数的所有节点和关系。每个节点包含的主要属性有：节点类型，代码，所在位置。每一条边代表节点之间的调用的先后顺序、数据和控制流的走向，如图 1 所示。其中红色线是数据流走向，蓝色线为控制流走向。A 节点为函数入口，E 节点为出口，中间三个节点分别为右侧代码的每一行。

1.2 API 序列提取

API 序列由 API 节点构成，API 节点为在参数和局部变量声明中所包含的全部类型及调用函数的名称[9]。将属性图中数据流和控制流的源头和汇合的节点命名为"源"和"汇"。以图 1 为例，"源"为节点 B，"汇"为节点 D。从图模型中观察得到，节点"源"和"汇"所在子树中包含函数参数、变量和调用，及全局和局部变量等信息。这些信息与软件安全操作所需条件相关。通过遍历代码属性图，标记处图中的"源"与"汇"，并在抽象语法树中找到标记函数所在节点。以该节点为根，提取出子树节点构成 API 序列。图 1 中 D 节点在抽象语法树中的子树如图 2 所示。

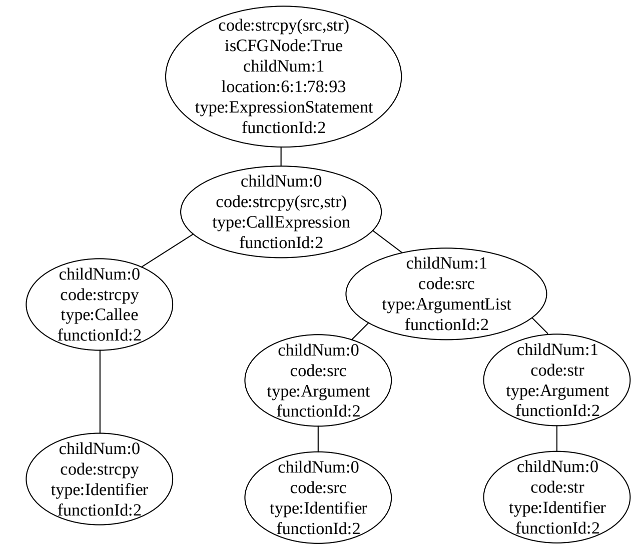


图 2 包含节点 D 的抽象语法树子树

Fig. 2 Abstract syntax tree subtree containing node D

该过程类似程序切片，从完整源代码的属性图中，提取出"源"与"汇"所在的子图的代码信息，构成 API 序列。所生成的序列包含"源"与"汇"的函数名称，使用的参数以及变量信息，条件语句以及节点之间的依赖关系。其中，条件语句为包含 if，for，while 等关键字的条件表达式。

1.3 量化 API 序列

通过上述方法得到的 API 序列为一组由表达式组成的具有抽象意义的值，无法直接作为机器学习的输入。因此，需要将序列嵌入向量空间，得到与之对应的数值向量，即样本特征。由于不同开发者的代码书写习惯不同，需要对函数名，参数名以及变量等信息进行归一化处理，以获得统一的形式。

首先，去掉 API 序列中的与漏洞信息无关的非 ASCII 字符；然后，将用户定义的变量以一对一的方式映射到符号名称（例如"ARG1""ARG2"），用户定义的函数和参数也以同样的方式进行映射（例如"FUN1""PAR1"）；最后，将返回值重新统一命名（例如"RET1""RET2"）。

得到归一化序列后，通过映射函数将其嵌入向量空间。此处引入"bag-of-word"模型，借鉴文本自然语言处理的方法将序列量化。对于自然语言来说，文本是由单词组成，通过统计关键词出现的频率即可得到其特征向量。而对于 API 序列来说，API 节点就是关键词。对每个开源软件，扫描程序得到所有函数集合 $X=\{x_1,x_2,\cdots,x_n\}$，生成每个源代码文件的 API 节点集合 A_i，并组合生成 API 词典 $A=\{A_1,A_2,\cdots,A_m\}$。映射函数 ϕ 可以表示为

$$\phi(x)=I(x,a)\cdot w_{a,A} \tag{1}$$

其中：$I(x,a)=\begin{cases}1 & \text{如果}x\text{包含API节点}a\\ 0 & \text{其他}\end{cases}$，$w_{a,A}$ 为术语频率逆文档频率（TF-IDF）的参数，用以降低 API 序列相似程度，消除常用 API 节点对于特征的影响，其表达式为

$$w_{a,A}=tf_{a,A}\times\log(N/df_a) \tag{2}$$

其中：$tf_{a,A}$ 为 API 节点 a 在 A 中出现次数和总节点数的比值，df_a 为节点 a 出现的文件数。

2 开源软件源漏洞检测模型

2.1 AE-KNN

量化得到的样本特征包含了源代码缺陷检测所需信息。

此外，本文还采用具有邻域发现的 KNN 算法来识别函数的邻域，获得程序上下文信息。考虑到本文样本特征维数较高，为提高性能，引入自动编码器（Autoencoder）来压缩样本特征，提出一种 AE-KNN 深度聚类模型，如图 3 所示。

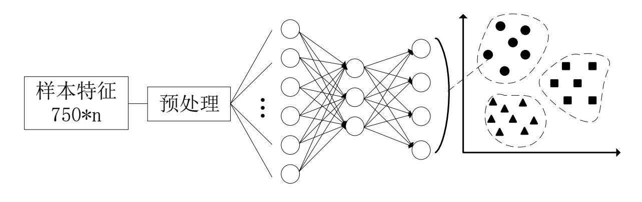


图 3　AE-KNN 模型

Fig. 3　AE-KNN model

该模型包括样本特征预处理，样本特征重构以及样本聚类三个部分。预处理用于除去噪声数据和重复数据，并进行数据交换使其转换为适合于数据分析的形式。本文采用 Z-score 规范化对样本特征进行预处理，假设样本容量为 n，样本中第 i 个特征的取值为 $x=\{x_{i1},x_{i2},\cdots,x_{in}\}$，规范化公式为

$$x_{ij}^{*}=\frac{x_{ij}-\bar{x}}{\sigma} \tag{3}$$

其中：$\bar{x}$ 和 σ 分别为该特征值的平均值和标准差。

样本特征重构部分对预处理后的特征进行挖掘，学习样本特征的深层抽象关系，并重构得到低维抽象特征。本文采用 Autoencoder 算法重构样本特征。Autoencoder 包括 encode 和 decode 两个过程，输入层、输出层及隐含层三个部分，encode 用于数据压缩，decode 用于数据重构。样本特征重构过程表示为

$$\begin{cases} y=f_{\theta}(x)=s(Wx+b) \\ z=g_{\theta'}(y)=s'(W'y+b') \end{cases} \tag{4}$$

其中：x 为输入样本特征，z 为重构样本特征，$f_{\theta}(\cdot)$ 为输入到隐含层的非线性函数，y 为中间结果，$g_{\theta'}(\cdot)$ 为隐含到输出层的非线性函数。W 和 W' 代表编码权重和解码权重，b 和 b' 分别为网络的偏移矢量，s 和 s' 为激活函数。该算法重构误差小，可以认为经过压缩后的低维特征几乎包含了输出数据的所有特征信息。

样本聚类部分将得到低维特征以“源”与“汇”所在的样本点为中心进行聚类。考虑到漏洞检测的重点在于代码片段的个体差异，因此实验采用余弦距离。假设 x 为待分类样本，c 为其中一类样本，d 为聚类中心，则算法可以表示为

$$y(x,c_j)=\sum_{d_i\in knn} sim(x,\ d_i)\,y(d_i,c_j) \tag{5}$$

其中：$sim(\cdot)$ 为 x 与 d_i 的余弦距离。

KNN 是一种 lazy-learning 算法，分类器依据 k 个样本中占优类别进行决策。这种几何表示能够识别“源”与“汇”的 k 个最近邻居。

2.2　漏洞检测

2.2.1 开源软件漏洞库

开源软件漏洞检测方法需要基于一定的漏洞集合，因此需要构建开源软件漏洞库。参照包括 CVE 漏洞数据库、美国国家漏洞库（NVD）等在内的知名漏洞库，并结合自身方法，构建了开源软件漏洞库。目前数据库收录了输入验证、缓冲区溢出、内存管理、API 误用等问题所涉及的漏洞，表 1 中显示了部分与以上问题相关的关键函数。

2.2.2 异常值计算

AE-KNN 模型可通过几何的方式来确定源代码中的漏洞。计算模型中样本的余弦距离用于邻近发现，可以快速发现相似的 API 使用模式，为漏洞检测奠定基础。计算各个类中每个样本的所占分数，用以推断“源”与“汇”邻居函数中的安全缺失，计算其异常分数。在漏洞库中查找匹配异常分数排名高的函数，得到源代码所含漏洞。

表 1　代码问题及对应关键函数

Table 1　Code problems and corresponding key functions

代码问题	关键函数
输入验证	insect，create，select，alter，update，exec，order，cookie，subject，system
缓冲区溢出	strcpy，strlen，strcat，strchr，scanf，sprintf，strerror，strcoll，sbumpc，malloc，recv，memcpy，fget，getpass
API 误用	cin，gets，fgets，getchat，getc，getpass，memcpy，malloc，getParameter
内存管理	malloc，calloc，realloc，alloca，free，new，delete，memcpy

根据 AE-KNN 的聚类结果，计算每个类中的各个函数的异常值。假设每个类的样本集合为 N，样本的向量表示为 $\vec{x}_i$，考虑每个类中的“源”与“汇”，构造正态模型为

$$\mu=\frac{1}{|N|}\sum_{\vec{x}_i\in N}\vec{x}_i \tag{6}$$

其中：μ 为质心向量，向量中每个值为邻居分数，数值分布为[0,1]，代表在进行样本邻近检查时，有百分之 μ 的函数返回了检查值。得到正态模型后，使用 $L-\infty$ 范数来计算出每个样本的异常分数为

$$f(x)=\|\mu-\vec{x}_i\|_{\infty} \tag{7}$$

对所有样本按照异常值进行排序，排名越高的样本所指向的代码段含有漏洞的几率越大。提取出高排名样本对应代码片段，并在开源软件漏洞库中匹配，最终得到被测软件的漏洞。

2.3　开源软件漏洞检测流程

根据前面的描述，本文提出的基于深度聚类的国家电网开源软件漏洞检测方法的具体流程如下：

a)根据开源软件源代码生成代码属性图，该代码属性图包含源代码数据、控制流信息，及代码嵌套结构。

b)遍历代码属性图，寻找数据流与控制流的“源”与“汇”节点，并提取出以该节点为跟的子树的关键 API 序列嵌入向量空间，得到特征向量。

c)采用 AE-KNN 模型对特征向量进行聚类分析，对得到的每一类计算出类中每个特征样本的异常值并排序，得到检测报告。

d)提取出报告中异常值排序高的样本函数，匹配漏洞库，得到函数中含有的漏洞，如图 4 所示。

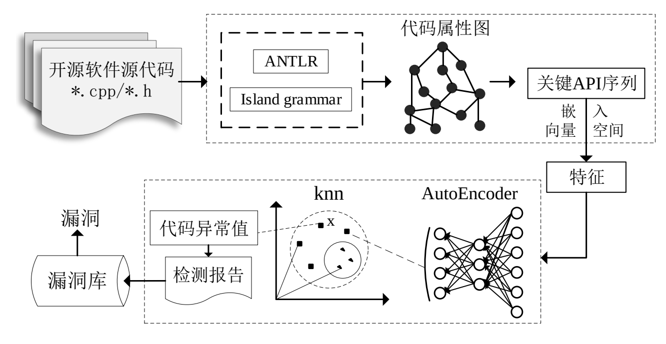


图 4　基于深度聚类开源软件漏洞检测模型

Fig. 4　Open source software vulnerability detection model based on deep clustering

3　算例分析

3.1　实验数据

统计国家电网公司自 2017 年 7 月到 2018 年 4 月的公司信息化项目采用的开源软件，结果显示项目共使用开源软件 104 个，占总数的 74.23%。本文选择两个使用较多的开源软件作为实验样本：

a)Redis。ANSI C 语言编写的日志型、Key-Value 数据库。2.8.2 以前以及 3.0.2 之前的 3.x 版本的 Redis 由于缺乏有效的认证，允许远程攻击者通过 eval 命令执行任意 Lua 字节码（CVE-2015-4335）。实验使用版本为 2.8.20。

b)MongoDB。C++语言编写的基于分布式文件存储的数据库。2.4.8 以前以及 2.6.8 之前的 2.6.x 版本的 MongoDB 允许远程攻击者通过 BSON 请求中的特殊 UTF-8 字符串导致拒绝服务（CVE-2015-1609）。实验使用版本为 2.6.7.

对于以上两个开源软件，根据上述方法构造其函数集合 X，通过分析每个“源”与“汇”所在类中的样本函数，得到函数的排名。值得注意的是，由于整个集合 X 中的函数都用于邻域选择，所得到的排名中存在具有漏洞的函数，也存在正常函数。

此外，为了验证不同漏洞类型的检测性能，根据美国国家标准与技术研究院官网提供的数据集，整理出 315 段代码作为实验集，其中正常代码 171 个，漏洞代码 144 个（输入验证 40 个，缓冲区溢出 43 个，API 误用 33 个，内存管理 28 个）。

3.2　实验及结果分析

3.2.1 参数设置

由于 AE-KNN 需要确定邻居数 k 的取值。计算不同 k 值下两个开源软件的平均聚类效果，并生成 ROC 曲线图，如图 5 所示。从图中可以看出当 k 取 25 时，获得的聚类效果最佳。故在实验中选定 k 的值为 25。

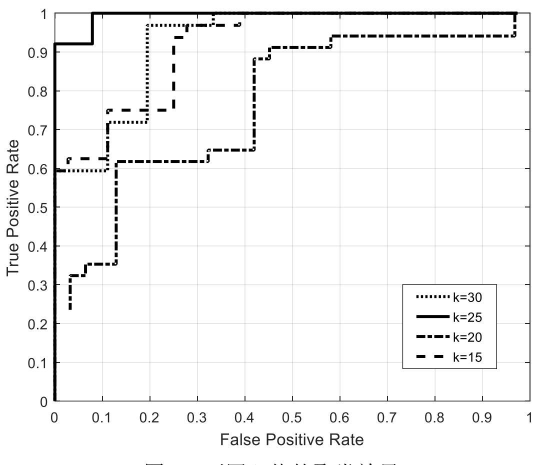


图 5　不同 k 值的聚类效果

Fig. 5　Clustering with different k

3.2.2 实验结果分析

对于 Redis，实验发现了 518 个“源”与“汇”，并对该 518 个“源”与“汇”进行邻近发现，并计算出函数的异常值，余弦距离以及分数并打印出来，如图 6 所示。从这 518 个结果中筛选出排名高的 6 个函数，如表 2 所示。

追溯这些函数，发现除表中第 5 个函数外，均与漏洞 CVE-2015-4335 相关，如表 3 所示。函数 luaV_execute 中的 case 语句 OP_FORLOOP 段中没有对参数进行类型检查，定义 idx 后就直接调用函数 luai_numadd()进行赋值。由于默认参数为 lua_Number 类型，导致了任意类型到 lua_Number 的混淆。恶意用户可将其他 case 语句的指令修改为 JMP 指令，来跳过语句中的类型检查，直接转入 OP_FORLOOP。此外，在函数 luaV_execute 中的另一个 case 语句 OP_CLOSURE 中，存在对闭包处理的 for 循环语句。其功能具体为在 CLOSURE 指令之后生成对应的 MOVE 指令，该指令的第二个参数为闭包变量的引用。正常情况下，此引用只能指向当前栈中的局部变量，但恶意用户可通过修改字节码，将其指向任意位置。在函数 luaD_precall 中，将创建的 C 语言闭包对象 CClosure 指向了函数指针。恶意用户覆写 CClosur→f，同时获取 System 地址并覆写至 fputs.got 后，即可利用该漏洞，并通过 eval 命令执行任意 Lua 字节码。

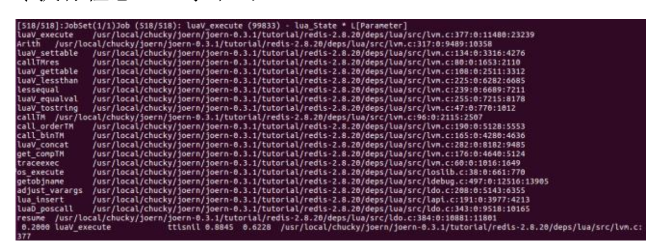


图 6　部分实验结果

Fig. 6　Partial experimental results

表 2　Redis 检测中六个高排名函数

Table 2　Six high ranking functions in Redis detection

异常值	分数	所在文件	函数名	行数
0.88	0.81	ldo.c	luaD_precall	307
0.80	0.75	lvm.c	luaF_findqupvvial	736
0.80	0.21	lvm.c	luaV_equalval	255
0.40	0.32	lgc.c	luaC_link	685
0.27	0.75	lvm.c	luaV_execute	377
0.27	0.48	lvm.c	luai_numadd	179

表 3　函数相关代码段

Table 3　Corresponding code segment of functions

函数	关键代码段
luaV_execute	case OP_FORLOOP: lua_Number idx=luai_numadd(nvalue(ra), step): case OP_CLOSURE: for(j=0; j<nup; j++,pc++){… ncl->l.upvals[j]=luaF_findupval(L, base + GETARG_B(*pc));}
luaD_precall	lua_assert(isLua(L->ci)); pc =L->savedpc; cl=&clvalue(L->ci-func)->l;

同样对于软件 MongoDB 进行检测，并提取出高排名的六个函数，如表 4 所示。这些函数中除第 4、5 两行外，均与漏洞 CVE-2015-1609 相关。在使用函数 validateBSON()时，对于原始输入 originalBuffer 只做了长度检查，就调用了 validateBSONIterative()来处理输入数据，如表 5 所示。此外，在 validateBSONIterative()调用的函数 readUTFString()，在读入了串的长度后，直接对变量 out 进行了赋值，没有对 sz 进行判断。由于缺少对输出值的检查， mongodb 的服务端无法验证一些畸形的 BSON 数据包，使得认证出现故障触发异常导致服务宕机。攻击者只需获得数据库访问权限，便可通过编写特定的数据包，即特定正则表达式，不断发送给数据库，导致服务器崩溃，造成拒绝服务攻击。

对于漏洞数据集，采用交叉验证测试该模型的准确性，将数据集划分为 5 个子集，每次选择一个子集作为测试集，其余作为训练集，交叉验证 5 次，实验结果取平均值。选择准确率（Accuracy），误报率（FPR），漏报率（Miss Rate）作为评价标准，计算公式为

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{8}$$

$$FPR = \frac{FP}{FP + TN} \tag{9}$$

$$MissRate = \frac{FN}{TP + FN} \tag{10}$$

其中：TP 为真实结果为正常样本，检测为正常样本；FN 为真实结果为正常样本，检测为漏洞样本；FP 为真实结果为漏洞样本，检测为正常样本；TN 为真实结果为漏洞样本，检测为漏洞样本。

表 4　MongoDB 检测中六个高排名函数

Table 4　Six high ranking functions in mongodb detection

异常值	分数	所在文件	函数名
0.84	0.76	bson_validate.cpp	validateBSON
0.80	0.69	bson_validate.cpp	ValidateBSONIterative
0.65	0.54	bson_validate.cpp	readUTF8String
0.50	0.47	bsonextract.cpp	bsonExtractIntegerFeildImpl
0.50	0.65	bsonextract.cpp	bsonExtractIntegerFeildWithDefaultIf
0.44	0.32	bson_validate.cpp	validateElementInfo

表 5　函数相关代码段

Table 5　Corresponding code segment of functions

函数	关键代码段

validateBSON	Status validateBSON(const char* originalBuffer, uint64_t maxLength) {
	if (maxLength < 5) {
	return Status(ErrorCodes::InvalidBSON, "bson data has to be at least 5 bytes");}
	Buffer buf(originalBuffer, maxLength);
	return validateBSONIterative(&buf); }

readUTF8String	if (!readNumber<int>(&sz)) return makeError("invalid bson", _idElem); if (out) {*out = StringData(_buffer + _position, sz); }

实验结果如表 6 所示，从表中可以看出本文提出的漏洞检测模型在以下 4 种漏洞代码上均能获得较高的检测准确率以及较低的漏报率。对于误报率，该模型在缓冲区溢出、API 误用及内存管理相关漏洞代码的表现较差，误报率较高。这可能是因为部分漏洞代码的关键函数有交集。

表 6　AE-KNN 模型实验结果

Table 6　Experiment results of AE-KNN

样本	准确率	漏报率	误报率
输入验证	92.89%	2.13%	9.5%
缓冲区溢出	93.64%	2.76%	11.6%
API 误用	93.03%	2.76%	13.93%
内存管理	93.17%	2.88%	15.71%
数据集	90.86%	12.22%	6.55%

为了进一步验证本文提出的漏洞检测方案的检测性能，在同一数据集下，将本文方法与其他漏洞检测算法进行比较，结果如表 7 所示。从表中可以看出，动态检测方案 VDiscover 的效果不如静态检测方案，而对于多种漏洞检测，本文提出的方案获能得较好的检测效果。

表 7　不同漏洞检测方案结果

Table 7　Results of different vulnerability detection scheme

模型	误报率	准确率
VulDeePecker[16]	5.7%	87.91%
Chucky[7]	-	89.78%
VDiscover[17]	16.34%	81.07%
AE-KNN	6.55%	90.86%

4　结束语

针对企业开源软件的安全性问题，本文提出基于深度聚类的源代码漏洞检测方法，利用代码图模型并引入自然文本语言分析技术来构造开源软件的样本特征，提出 AE-KNN 模型并进行聚类分析，计算每个样本的异常值并排序，得到关键代码段并通过匹配漏洞库检测出代码段中的漏洞。该检测模型的局限性在于高排名的函数中，可能存在正常函数。在未来的研究中，将着眼于提高模型的精确度，实现完全自动化地检测国家电网开源软件源代码中的漏洞，包括已知漏洞和零日漏洞，同时扩展到其他语言编写的开源软件。

参考文献：

[1] 蔡军，邹鹏，杨尚飞. 软件漏洞分析中的脆弱点定位方法 [J]. 国防科技大学学报, 2015, 37 (5): 141-148. (Cai Jun, Zou Peng, Yang Shang fei. Vulnerable spots localization methods for software vulnerability analysis [J]. Journal of National University of Defense Technology, 2015, 37 (5): 141-148.)

[2] 李舟军，张俊贤，廖湘科，等. 软件安全漏洞检测技术 [J]. 计算机学报，2015, 38 (4): 717-732. (Li Zhoujun, Zhang Junxian, Liao Xiangke, *et al.* Survey of software vulnerability detection techniques [J]. Chinese Journal of Computers, 2015, 38 (4): 717-732.)

[3] 徐威扬，李尧，唐勇，等. 一种跨指令架构二进制漏洞搜索技术研究 [J]. 信息网络安全, 2017 (9): 21-25. (Xu Weiyang, Li Yao, Tang Yong, *et al.* Research on cross-architecture vulnerabilities searching in binary executables [J]. Netinfo Security, 2017 (9): 21-25.)

[4] Li Zhen, Zou Deqing, Xu Shouhuai, et al. VulDeePecker: a deep learning-based system for vulnerability detection [EB/OL]. (2018-01-05). https://arxiv.org/pdf/1801.01681.pdf.

[5] Perl H, Dechand S, Smith M, *et al.* VCCFinder: finding potential vulnerabilities in open-source projects to assist code audits [C]//Proc of ACM SIGSAC Conference on Computer and Communications Security. New York：ACM Press, 2015: 426-437.

[6] Walden J, Stuckman J, Scandariato R. Predicting vulnerable components: software metrics vs text mining [C]//Proc of IEEE, International Symposium on Software Reliability Engineering. Washington DC: IEEE Computer Society, 2014: 23-33.

[7] 危胜军，何涛，胡昌振，等. 基于组件依赖图的软件安全漏洞预测方法 [J]. 北京理工大学学报, 2018, 38 (5): 525-530 . (Wei Shengjun, He Tao, Hu Changzhen, *et al.* Predicting software security vulnerabilities with component dependency graphs [J]. Transactions of Beijing Institute of Technology, 2018, 38 (5): 525-530.)

[8] Yamaguchi F, Wressnegger C, Gascon H, *et al.* Chucky: exposing missing checks in source code for vulnerability discovery [C]//Proc of ACM Conference on Computer and Communications Security.New York: ACM Press, 2013: 499-510.

[9] Shar L K. Predicting common web application vulnerabilities from input validation and sanitization code patterns [C]//Proc of IEEE/ACM International Conference on Automated Software Engineering. New York: ACM Press, 2012: 310-313.

[10] Scandariato R, Walden J, Hovsepyan A, *et al.* Predicting Vulnerable Software Components via Text Mining [J]. IEEE Trans on Software Engineering, 2014, 40 (10): 993-1006.

[11] Pang Y, Xue X, Namin A S. Predicting vulnerable software components through n-gram analysis and statistical feature selection [C]//Proc of IEEE International Conference on Machine Learning and Applications.

IEEE, 2015: 543-548.

[12] 杨琪. 基于深度学习的聚类关键技术研究 [D]. 成都：西南交通大学, 2016.(Yang Qi. Research on key technologies of clustering based on deep learning[D].Chengdu:Southwest Jiaotong University,2016.)

[13] Joern[DB/OL]. https://github.com/octopus-platform/joern.

[14] Rodriguez M A, Neubauer P. The graph traversal pattern [J]. Graph Data Management Techniques & Applications, 2010. https://arxiv.org/abs/1004.1001.

[15] 缪旭东，王永春，曹星辰，等. 基于模式匹配的安全漏洞检测方法 [J]. 计算机科学，2017，44 (4)：109-113. (Miao Xudong, Wang Yongchun, Cao Xingchen, *et al.* Detection approach for security vulnerability based on pattern matching [J]. Computer Science, 2017, 44 (4): 109-113.)

[16] 原子，于莉莉，刘超. 引入缺陷的细粒度软件变更识别方法 [J]. 北京航空航天大学学报, 2014, 40 (9): 1231-1238. (Yuan Zi, Yu Lili, Liu Chao. Identification method for defect-introducing fine-grained software changes [J]. Journal of Beijing University of Aeronautics and Astronautics, 2014, 40 (9): 1231-1238.)

[17] Grieco G, Grinblat G L, Uzal L, *et al.* Toward Large-Scale Vulnerability Discovery using Machine Learning [C]//Proc of ACM Conference on Data and Application Security and Privacy. New York:ACM Press, 2016: 85-96.